Защита от **OP**[1]-инжекта.

OP-инжект(исполнение **OP** кода) основан на использовании серии блоков кода(гаджеты), переходы между блоками выполняются через инструкции возврата(**ROP**: *Ret*), либо через ветвления(**JOP, COP**). Блоки расположены внутри тела легитимных функций, обычно ближе к концу процедур. Начало блоков может коррелировать с **CFG**, первая инструкция блока расположена выше оригинальной инструкции, но в пределах её размера.

Процедура является целостным обьектом, имеющим точки входа и выхода. Все части процедуры связаны в **CFG**, произвольная передача управления на тело процедуры нарушает **CFG**. Всякая передача управления не на начало процедуры, за исключением процедурного возврата (не нарушающего **CFG)** является событием **OP**. При процедурных вызовах из процедуры на стек загружаются локальные точки входа, передача управления на которые возможна только после их загрузки(процедурного ветвления) и до возврата на эти точки. Таким образом для блокировки **OP** необходимо обнаружить событие передачи управления на адрес, который не является **EP**[2]. В частности **OP**-код может быть расположен не в пределах легитимной процедуры, а в области данных, которая доступна к исполнению. Обнаружить передачу управления на **EP** и заблокировать **OP** можно следующими способами:

1. Удалить из памяти тело процедуры и записать в точку входа ветвление(либо шлюз - *атом*[3]) на копию процедуры. Копия может быть линейной - всего блока с кодом(секции модуля), либо выделенным и изменённым кодом. Тогда исходное тело будет не исполняемым(за исключением точки входа), например заполненным инструкциями, приводящими к исключению. Как частный случай может быть запрещён доступ к области памяти с телом процедуры(**NX**)[3][2].

```
4B30B020 RtlFillMemoryUlong  $  57           push edi
4B30B021                     .  8B7C24 08    mov edi,dword ptr ss:[esp+8]
4B30B025                     .  8B4C24 0C    mov ecx,dword ptr ss:[esp+C]
4B30B029                     .  8B4424 10    mov eax,dword ptr ss:[esp+10]
4B30B02D                     .  C1E9 02      shr ecx,2
4B30B030                     .  F3:AB        rep stos dword ptr es:[edi]
4B30B032                     .  5F           pop edi
4B30B033                     .  C2 0C00      ret 0C
```

```
4B30B020 RtlFillMemoryUlong  - E9 593521B5    jmp 0051E57E
4B30B025                       CC             int3
4B30B026                       CC             int3
4B30B027                       CC             int3
4B30B028                       CC             int3
4B30B029                       CC             int3
```

```
0051E57E                       57             push edi
0051E57F                       8B7C24 08      mov edi,dword ptr ss:[esp+8]
0051E583                       8B4C24 0C      mov ecx,dword ptr ss:[esp+C]
0051E587                       8B4424 10      mov eax,dword ptr ss:[esp+10]
0051E58B                       C1E9 02        shr ecx,2
0051E58E                       F3:AB          rep stos dword ptr es:[edi]
0051E590                       5F             pop edi
0051E591                       C2 0C00        ret 0C
```

2. Изменить тело процедуры(**CCFIR**[1]). Тогда оно будет отличиться от оригинального и передача управления на гаджет приведёт к ошибке. Как и в способе 1 фактически это скрытие реальных частей тела процедуры, блокировка **OP** в этом случае является следствием передачи управления на некорректный адрес.

3. При возврате из процедуры проверить(**CFI**) адрес возврата, который сохранён при вызове процедуры. Частным случаем является защита адреса возврата от перезаписи (***COOKIE***). Для этого необходима поддержка компилятора(**ms-RFG**[i]) или **CPU**(**i-**

1 ”Oriented Programming”.
2 Точка входа.
3 Не доступная для чтения тела процедуры, вызываемая через специальный шлюз.

CET[ii]**:** *ENDBRANCH*).

4. Использовать анализ **CFG**/**DFG** – динамические способы.

Для реализации первых двух способов необходимо знать **EP** всех процедур в приложении. По умолчанию **PE**[iii] формат такой информации не содержит. Список **EP** процедур является расширением **PE** в виде **CFT** таблицы(*GuardCFFunctionTable*). Эта таблица используется для валидации указателей на процедуры при косвенных ветвлениях(**ms-CFG**, частный случай 5). В **CFT** описаны процедуры(**EP**), вызов которых происходит косвенным путём(на которые есть ссылки), либо которые экспортируются(переменные исключаются).

При загрузке модуля по произвольному адресу выполняется настройка указателей, расположенных в модуле. Ссылки(релоки) на указатели описаны в таблице базовых поправок. Указатель на процедуру(**EP**), вызываемую косвенным ветвлением, должен быть описан в таблице релокаций(в модуле должен быть релок, содержащий данный указатель). Это частный случай #[2] и только для **x86**. Модуль из которого запускается процесс может может не содержать релоков, так как при его загрузке адресное пространство не содержит проекций и корреляции адресов не возникает. Но такой модуль может содержать релоки для адресной рандомизации(**ASLR**).

В 64-х битном режиме используется *относительная адресация(***RIP***)*. Эффективный адрес вычисляется сложением текущего адреса инструкции и смещения:

Table 2-7. RIP-Relative Addressing

ModR/M and SIB Sub-field Encodings		Compatibility Mode Operation	64-bit Mode Operation	Additional Implications in 64-bit mode
ModR/M Byte	mod = 00	Disp32	RIP + Disp32	Must use SIB form with normal (zero-based) displacement addressing
	r/m = 101 (none)			
SIB Byte	base = 101 (none)	if mod = 00, Disp32	Same as legacy	None
	index = 100 (none)			
	scale = 0, 1, 2, 4			

Так как смещение на адресуемые данные постоянно в пределах модуля, то загрузка модуля на другие адреса не требует использования релоков для кода. В этом случае релоки нужны для ссылок в области данных. Релоки не содержат информации, позволяющей узнать что по указателю расположен код(не данные). Эта информация может быть получена на основе **EP** в **CFT**. Релок показывает что данные по ссылке являются указателем. При отсутствии релоков обнаружить указатель можно только по **RIP**-адресации.

Получение всех **EP** состоит из 3-х этапов(статический анализ):

1. Получение всех указателей. При отсутствии релоков(**x64**) - поиск всех **RIP**-адресаций(каждая является ссылкой). Возможно линейное сканирование памяти(**MODRM**[4]**(RIP)** и **EA** в пределах модуля) или создание и анализ **CFG**.

2. Выделение указателей на код. Для этого необходима проверка валидности **CFG** и прочие проверки. Например *эффективный адрес*(**MODRM**) должен быть корректен:

- **[R*4]**	База не равна нулю.
- **[R + R*S]**	База не равна индексу.
- **([R32] + R8)**	Операнд не является частью базы.

4 Формат кодировки эффективного адреса(**EA**).

- **[DISP32]**	Релок только на смещении.
- **[DISP32], IMM32**	Релок на DISP32 или IMM32.

3. Анализ кода по указателям для выделения **EP** вложенных процедур. Код должен быть описан через **CFG** и выделены все **EP**, вызываемые через относительные процедурные ветвления.

Для примера взят модуль **ntdll**(10.0.10586). Исходными точками для формирования **CFG** служат **CFT**-указатели. Покрывается почти весь код модуля, за исключением небольшого числа процедур, связанных с обработкой исключений.

Экспорт	1754
CFT	1952
Процедур	5245

Наследование указателя(**PFG**).

Возможна вариация **OP**(**COP/JOP**), когда передача управления между **OP**-гаджетами происходит не через инструкции возврата, а через косвенные ветвления. Разрешение передачи управления только на **EP** не заблокирует **OP**, который выполняется в виде серии косвенных вызовов **EP**. Передача управления на **EP**, ссылка[5] на которую отсутствует, за исключением относительного процедурного ветвления, является событием **OP**. Указатель на код не может вычисляться. До вызова **EP** должна произойти **R/X-DF**[6] по адресу ссылки, например:

```
mov R,[P]     ; R-DF по ссылке P.
Call R        ; Ветвление по ссылке [P].
```

Другой пример:

```
push P        ; X-DF по адресу P в инструкции.
ret           ; Ветвление на P.
```

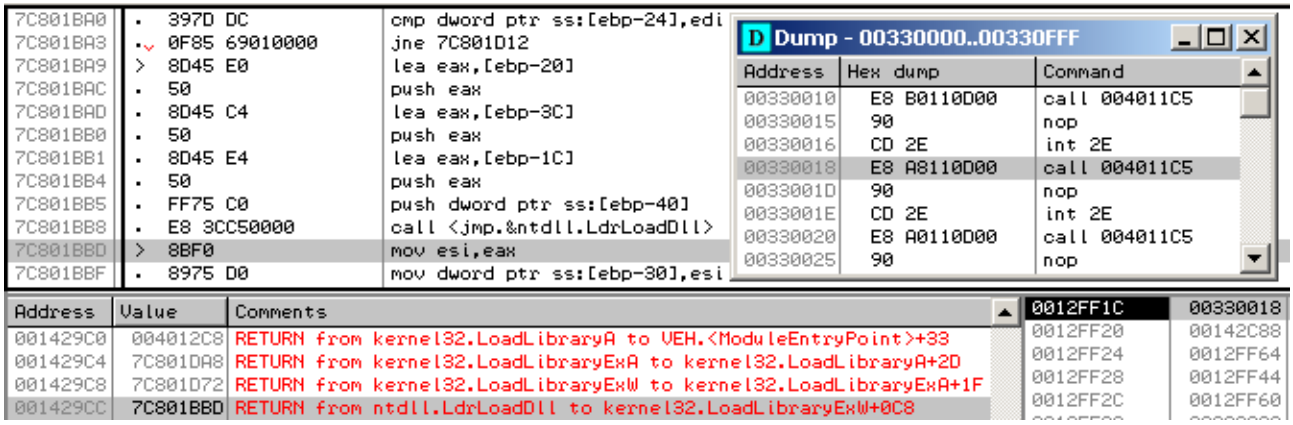
В данном случае при использовании теневого стека(*Shadow Stack*[7]) будет обнаружена **OP** на инструкции **Ret**(соответствующее процедурное ветвление исполнено не было и теневой стек не содержит адреса возврата), например для ***CET***:

```
Operation
(* Near return *)
IF instruction = near return
    THEN;
        IF OperandSize = 32
            THEN
                IF top 4 bytes of stack not within stack limits
                    THEN #SS(0); FI;
                EIP <- Pop();
                    IF ShadowStackEnabled(CPL)
                        tempSsEIP = PopShadowStack4B();
                        IF EIP != TempSsEIP
                            THEN #CP(NEAR_RET); FI;
                    FI;
```

5 Ссылка — указатель расположенный в памяти.
6 Выборка данных, при исполнении(**X-DF**) из тела инструкции или при адресации данных(**R/W-DF**).
7 Теневой стек используется для хранения адресов возврата. При процедурном ветвлении в него сохраняется адрес возврата, который проверяется при возврате из процедуры. Используется в **i-CET** и **ms-RFG**.

Такая передача управления не является исполнением **OP**, так как имеется указатель в инструкции и он выбран(**X-DF**).

Обработка относительных процедурных ветвлений происходит через механизм **TCM**[iv]**.** Точка входа, которая является адресом возврата из вызываемой процедуры локальна для ветвления и существует только для него. Как следствие произвольная передача управления на эту точку является событием **OP**. При процедурном ветвления в стек сохраняется адрес шлюза(ключ); адрес возврата сохраняется в **TLS**[8]. Возврат из процедуры выполняется на шлюз, в котором выбирается адрес возврата на основе ключа.



– Пример обработки серии вложенных ветвлений(слева показан массив реальных адресов возврата).

В более общем представлении перед **DF** по указателю последний должен быть сформирован, для него должен иметься источник. Это может быть указатель, расположенный в инструкции или в памяти(ссылка). Во втором случае должна произойти **DF** по адресу ссылки и так же должен иметься источник адреса ссылки. Подобные зависимости описываются через **DFG**. Часть **DFG**, которая описывает поток данных, формирующих указатель, обозначим как **PFG**. Тогда можно определить **OP** атаку как внесение данных в **PFG:** указатель на который происходит передача управления является некоторой функцией входных данных. В работе # используется понятие ***DUI***[9].

```
mov Ra,P    ; Ra <- P
mov Rb,Ra   ; Rb <- Ra
add Rb,Rc   ; Rb + Rc: В PFG вводятся данные.
```

Для реализации **PFG** необходимо помечать каждый источник данных(регистр или адрес памяти) маркером(далее '**L**') , показывающим что данные являются указателем. Таким образом маркер(указатель) наследуется между источниками данных и выбирается(**DF**) в начале наследования.

8 Поточное хранилище данных.
9 *Dereference Under the Influence.*

```
mov R,P          ; R.L
mov [Ma],R       ; Ma.L
mov [Mb],[Ma]    ; Mb.L
Call [Mb]        ; Ветвление возможно если Mb.L
```

- указатель наследуется в **PFG**: R -> Ma -> Mb. Исходный указатель(P) статический, он расположен в инструкции.

Имплементация **PFG**.

Могут быть выделены три основные проблемы: разложения, смещения и отличия.

1. Разложение. В **PFG** указатель может быть разложен на части, например при побайтном копировании блока данных, когда произойдёт **DF** по ссылке с указателем, будет загружена в приёмник его часть(за одну инструкцию), таким образом приёмник не будет содержать указатель. Далее указатель сформируется при дальнейшем копировании остальной его части. В данном случае имеется наследование указателя по источнику и приёмнику данных в **DFG**, не зависимо от промежуточной операции(функции) со значением указателя. Так например указатель может шифроваться для скрытия или защиты и восстанавливаться перед обращением к нему, а промежуточная функция может быт не известна. Для решения данной проблемы должен быть введён дополнительный маркер, который покажет что указатель наследован, но не сформирован('**P**'). При **DF** по указателю, который помечен **P** должно быть проверено значение указателя(адреса).

Накопление точек входа.

Образу модуля или его кодовой секции сопоставляется битовая карта, каждый байт образа описывается двумя битами карты. Это два маркера, первый(**L**) помечает адрес как имеющий ссылку, второй(**P**) помечает адрес как **EP**. При обнаружении в **IF**[10] выборки(**R/X-DF**) данных по ссылке устанавливается маркер **L** для указателя по ссылке. Когда происходит ветвление, проверяется маркер **L** и если он установлен, то взводится маркер **P** и ветвление разрешается. Иначе это выполнение **OP**. При наличии **CFT** устанавливаются соответствующие маркеры **P**, но всё равно выполняется проверка **L**. Таким образом соблюдаются условия #[4] и нет необходимости выполнять аналитическое получение **EP**.

```
005BA890  $  60                    pushad
005BA891  .  A1 30C35C00           mov eax,dword ptr ds:[<&kernel32.VirtualFree>]
005BA896  .  C705 30C35C00 ACA85B00 mov dword ptr ds:[<&kernel32.VirtualFree>],005BA8AC
005BA8A0  .  A3 10EE5C00           mov dword ptr ds:[5CEE10],eax
005BA8A5  .  61                    popad
005BA8A6  .  68 98A65B00           push 005BA698
005BA8AB  .  C3                    ret
```

Рассмотрим примеры, *PE Explorer*:

```
005BA890 <ModuleEntryPoint>  $  60                    pushad
005BA891                     .  A1 30C35C00           mov eax,dword ptr ds:[<&kernel32.VirtualFree>]
005BA896                     .  C705 30C35C00 ACA85B00 mov dword ptr ds:[<&kernel32.VirtualFree>],005BA8AC
005BA8A0                     .  A3 10EE5C00           mov dword ptr ds:[5CEE10],eax
005BA8A5                     .  61                    popad
005BA8A6                     .  68 98A65B00           push 005BA698
005BA8AB                     .  C3                    ret
```

10 Поток инструкций.

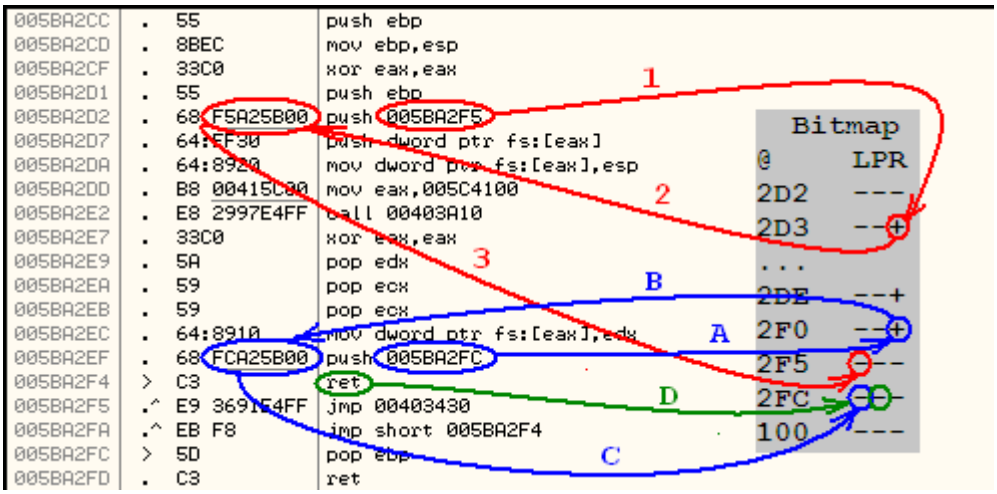
- В **IF** загружаются ссылки 5BA8AC и 5BA898, для указателей по которым устанавливаются **L**-маркеры. Когда происходит передача управления(*Ret*) на адрес 5BA698 проверяется маркер **L**. Так как он установлен, процедура 5BA698 помечается маркером **P** и исполнение разрешается. Аналогичная проверка выполняется при ветвлении на 5BA8AC.

Ссылки могут быть расположены не в теле инструкций(смещение), а в области данных. Для обнаружения таких ссылок необходимо соответственно отслеживать **R-DF**:

```
00431093   .  A3 70B85B00   mov dword ptr ds:[5BB870],eax
00431098   .  68 60B85B00   push 005BB860
0043109D   .  E8 A650FDFF   call <jmp.&user32.RegisterClassA>
004310A2   .  66:85C0       test ax,ax
004310A5   .v 75 21         jnz short 004310C8

005BB860   00000000
005BB864   00405D90 Entry address
005BB868   00000000
```

- Указатель на процедуру расположен в области данных(5BB864 помечен маркером **R**). При **R-DF** по ссылке 5BB864 читаемый указатель помечается маркером **L**. При наличии релокаций должна выполняться проверка релока для ссылки. Для **RIP** ссылки могут быть накоплены. Для этого может быть использован третий маркер(**R**) в битовой карте(третье двух битное значение), так как релок не может располагаться на **EP**:



- При **IF** по адресу 5BA2D2(загрузка **SEH**[11]) проверяется релок для ссылки [5BA2D3] в карте(1). Если ссылка помечена **R**, то выбирается(2) указатель [5BA2D3] -> 5BA2F5 и он помечается **L**(3). Во втором варианте происходит аналогичное(ABC). При возникновении исключения управление на **SEH**(5BA2F5) передаётся через косвенное процедурное ветвление. При этом проверяется маркер **L** и так как он установлен, то ветвление разрешается(**P**). Для второго варианта управление передаётся(D) через *Ret* – проверяется маркер **L** и устанавливается **P**. Для **R-DF** проверка выполняется аналогично.

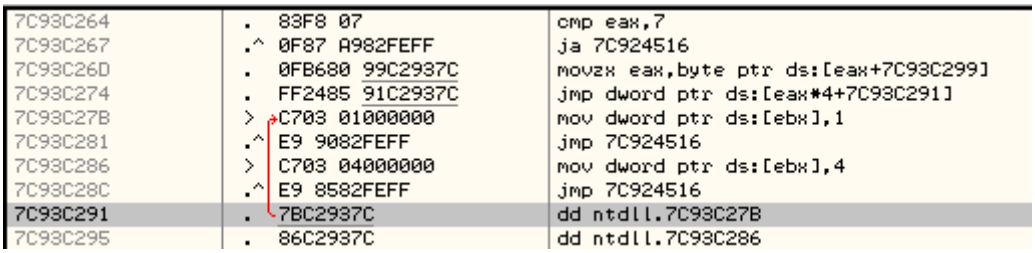
Для *косвенных* ветвлений(**RX-DF**) проверяется маркер **R** для **EA**. Если он установлен(**DF** по ссылке), то маркер **L** для указателя по ссылке не устанавливается, указатель помечается **P** и ветвление разрешается. Если **R** сброшен, то проверяется **L**. В противном случае установка **L** дала бы возможность выполнения кода по ссылке без её загрузки.

```
00496034   .  8B00       mov eax,dword ptr ds:[eax]
00496036   .  8D55 F4     lea edx,dword ptr ss:[ebp-C]
00496039   .  FF10        call dword ptr ds:[eax]      pexplore.004963EC
ds:[00495800]=004963EC (pexplore.004963EC)
00495800 004963EC pexplore.004963EC
```

11 Структурная обработка исключений.

- **EA**(495800) является ссылкой и помечен маркером **R**. Указатель 4963EC помечается **P** и ветвление разрешается. В противном случае(**!R**) для указателя проверяется маркер **L**.

При *косвенных индексируемых* ветвлениях выполняется выборка(**RX-DF**) только одного указателя из массива:



- При **IF** ветвления 7C93C274 вычисляется **EA** и проверяется маркер **R** для него. Так как ссылки 7C93C291/95 помечены маркером **R**, то указатель помечается **P**. Ссылки в массиве содержат не корректные указатели(**!L**), до такой же **RX-DF** по ним.

Алгоритм.

```
InstructionFetch:
        if ReadFetch                            ; R-DF
                M = LinearAddressOfFetch        ; Вычисляем LA DF.
                if SizeOfFetch() = 32           ; Выбирается указатель.
                        Ptr = [M]               ; Указатель по ссылке.
                        if Linear               ; Линейная инструкция.
                                if Bit{M}.R     ; LA является ссылкой.
                                        Bit{Ptr}.L = 1 ; Помечаем указатель как загруженный.
                                fi
                        else                    ; Косвенное ветвление.
                                if Bit{M}.R     ; Через ссылку.
                                        Bit{Ptr}.P     ; Разрешаем.
                                else
Access:                                 if !Bit{Ptr}.L ; Указатель не загружен.
                                                Break  ; OP — блокировка.
                                        fi
                                fi
                        fi
                fi
        elseif WriteFetch
                M = LinearAddressOfFetch        ; Вычисляем LA DF.
                if SizeOfFetch() = 32           ;
                        Ptr = Src               ;
                        > Access

        elseif Linear                           ; X-DF, линейная инструкция.
                if Operand32                    ; Имеется операнд.
                        M = AddressOfOperand    ;
                        Ptr = [M]               ; Значение операнда.
                        if Bit{M}.R             ; Операнд является указателем.
                                Bit{Ptr}.L = 1  ; Помечаем указатель как загруженный.
                        fi
                fi
        else
```

```
        if RET                          ;
                Ptr = [ESP]             ; Извлекаем указатель из стека.
                > Access                ;
        elseif INDIR CALL/JMP           ; Регистровое ветвление(MODR/M.Mod = 3).
                Ptr = FlowAddress       ; Адрес ветви.
                > Access                ;
        elseif REL CALL                 ; Относительное ветвление.
                SetTcmKey               ; TCM обработка.
        fi
fi
Continue
```

[3] DYPE.
[2] Combating the Advanced Memory Exploitation Techniques
[1] Practical Control Flow Integrity & Randomization for Binary Executables
i
ii Control-flow Enforcement Technology Preview

iii Microsoft Portable Executable and Common Object File Format Specification

iv Thread Call Map.